EXIST

EXpression of Information based on logic and Set Theory

Graphical Form Version 0.5

Matthew West - Shell Services International

Some Contributors
(whether they know it or not)

EXIST

- Chris Angus
- Ian Bailey
- Allison Barnard-Feeney
- Bill Burkett
- Bill Danner
- Julian Fowler
- Chris Partridge
- Dave Price
- Donald Sanderson
- Phil Spiby
- Bernd Wenzel
- Rob Whitesell

Introduction

EXIST

- Purpose
  - To provide the basis for integrating data from different data models potentially with different purposes and using different modelling languages.
- Forms of the Language
  - Lexical
  - XML
  - Graphical (this form)
  - Meta-model
  - Interface Specification

Key Capabilities

EXIST

- Able to support a model with different levels of abstraction
- Able to integrate models from different modelling languages
- Able to map between different models
- Able to make negative statements
- Able to be extended
- Sound basis on fundamental axioms only, i.e. logic and set theory
- Multiple equivalent forms

Language Elements

EXIST

- Base Elements
  - Named Objects
    - Names, namespaces, variables
  - Structures
    - Sets, tuples
- Set Membership
- Relations
  - General relations, functions and operations
- Operators

Object

EXIST

Test/aaa

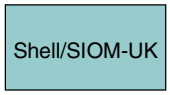
Definition: Set or an Individual.

The set or individual is represented by a sign, in this case a rectangle. A set or individual is not necessarily uniquely represented by a single symbol.

A sign for a set or individual may include a name by which it is known. All names **must** be in a name space, in this case Test.

Name Spaces

EXIST



Definition: set of names where each name refers to only one object within the set.

Note: More than one name can refer to the same object.

A Name Space can have a name. This can be within another name space.

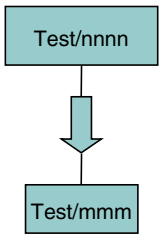
A "/" or a is used to separate a name from its name space.

E.g. Shell/SIOM-UK, (this does not indicate that SIOM-UK is a part of Shell)

EXIST uses the Name Space "$" for key words, where $==EXIST/

Tuples

EXIST



Definition: two or more elements that are ordered.

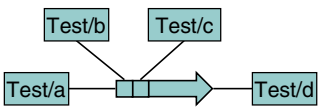
A tuple's identity is defined by its elements and their order - in other words, <Test/a, Test/b> always represents the same tuple as <Test/a, Test/b> .

An element may be repeated in a tuple.

A tuple may not refer to itself.

There is only one tuple <Test/nnnn, Test/mmm>

The sign for a binary tuple is a block arrow (order tail to head, e.g. <Test/nnnn, Test/mmm>).



An n-tuple e.g. <Test/a, Test/b, Test/c, Test/d>

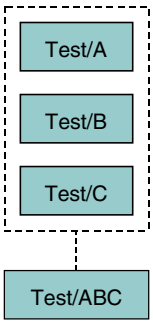
Notes on the Tuples

EXIST

- Tuples are NOT pointers.
- A tuple is NOT equivalent to a relationship in an entity relationship model.
- For 2 Things Test/A, Test/B there are exactly 4 tuples: <Test/A, Test/A>, <Test/A, Test/B>, <Test/B, Test/A>, <Test/B, Test/B>
- A tuple has no meaning on its own. It is given meaning by the sets that it is a member of.

Sets

EXIST



Definition: number or collection of things

A set's identity is defined by its membership (i.e. two sets with the same members are the same set)

A Thing cannot have repeated membership of a set.

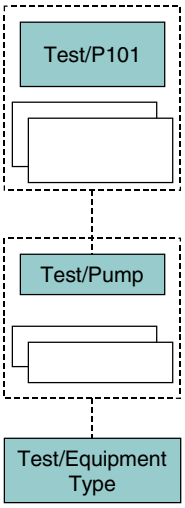
A set 's membership (extent) is symbolised by the attached dotted box, {Test/A, Test/B, Test/C}.

Unlike a tuple, a set is not ordered, so {Test/A, Test/B, Test/C} is the same set as {Test/C, Test/B, Test/A}.

The set is linked to its extent through an identity tuple e.g. [Test/ABC == {Test/A,Test/B,Test/C}]

Set Membership

EXIST



When not all the members of the set are shown, then a stacked unnamed sign is shown to indicate that there are other members.

[Test/Pump == {Test/P101, …}]

In the text version, the dots indicate the existence of other members.

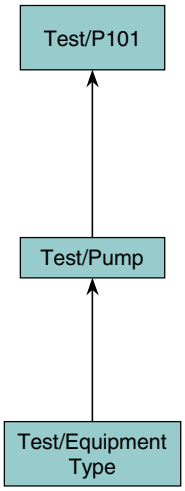
Set/Class membership is non-transitive. When one set is a member of another, this is indicated by the solid box representing the whole set being in the extent of the set it is a member of.

Thus whilst P101 is a member of the set Pumps, and Pumps is a member of the set Equipment Types, P101 is not a member of Equipment Types.

[Test/Equipment Types == {Test/Pump, …}]

Set Membership

EXIST



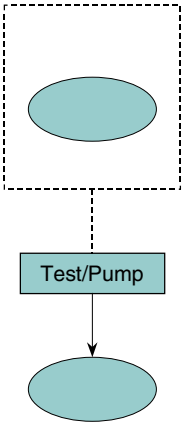
When it is only desired to indicate that one thing is a member of a set or class, then an arrow is used graphically, going from class to member, and lexically a ":" is used. This is just a simplified form of the previous notation.

[Test/Pump: Test/P101],

[Test/Equipment Types: Test/Pump]

Unbound Variables

EXIST

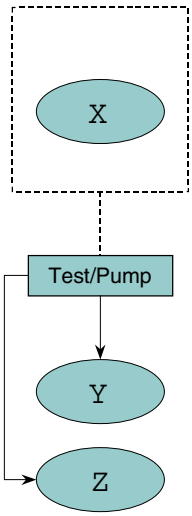


An unbound variable is indicated by an oval without a name with its membership of a set indicated either by a set-extent sign, or by a set membership arrow. The lexical equivalent is:

%Test/Pump%

Bound Variables

EXIST



A bound variable is indicated by an oval with a name. Its membership of a set is indicated either by a set-extent sign, or by a set membership arrow. The lexical equivalent is:

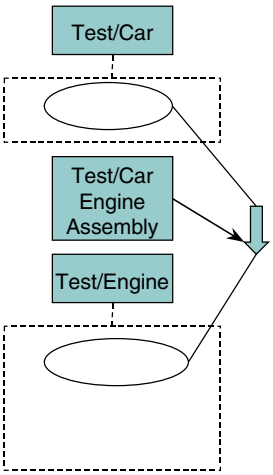
Test/Pump == {%X }

Test/Pump: %Y

A bound variable necessarily refers to the same thing in different references to the variable. The type over which the variable can range is also indicated.

Example of use of Variables

EXIST

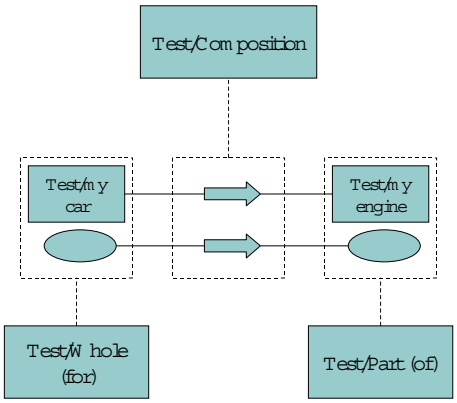


As an example consider the definition of the types allowed in a car engine assembly tuple.

[Test/Car Engine Assembly: <%Test/Car%, %Test/Engine%>]

Relations

EXIST

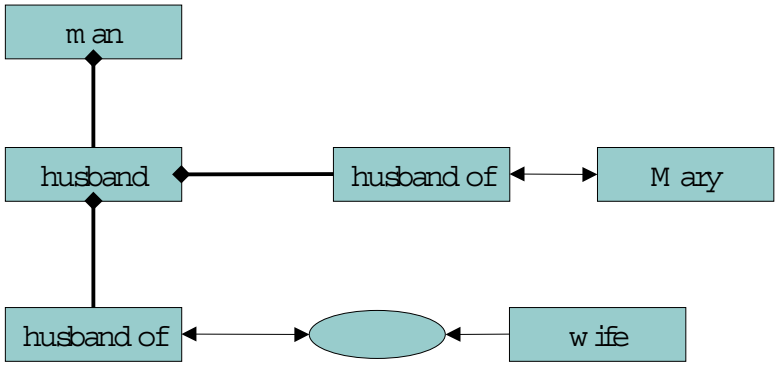


A Relation is a classification of a tuple. The roles played by the elements of the tuple are indicated by defining a constraint on the membership of the elements of the tuple through unbound variables.

Functions are relations which obey particular constraints, in that the mapping form the domain to the image is unique.

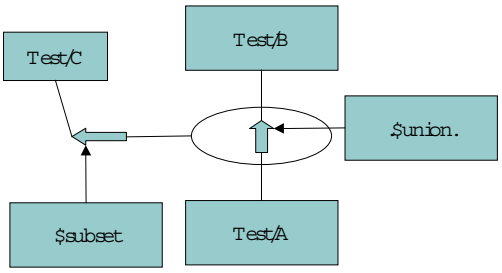
Complex Relations

EXIST



Operator

EXIST

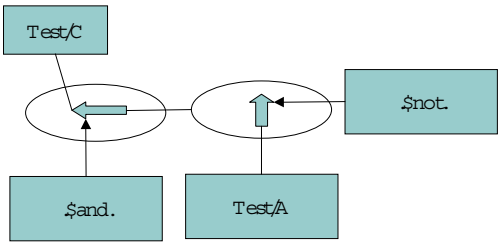


An Operator is a way of representing a function with one or two inputs and one output.

The Ellipse signifies the result of the operation, which can participate in other tuples.

Unary Operator

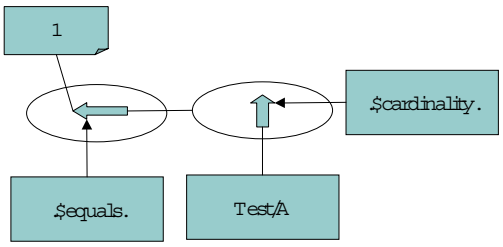
EXIST



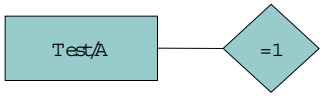
A unary operator such as .$not. Uses a tuple with only one element.

Cardinality

EXIST



Cardinality is formally represented as an equation. Test/A is the set whose cardinality (number of members) is being defined.



Since cardinality is a frequent construct, a shorthand form is provided.

Literals

EXIST

Definition: encoding which can be held in a computer in a (presumably) binary format

The Literal &'SIOM-UK'&. This refers to the Literal SIOM-UK rather than what the name represents.

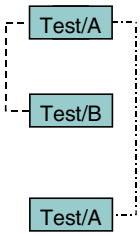
The possible delimiters are ', '', or &' and '&.

Literals are members of classes that denote their type e.g. ASCII, Unicode IEEE numbers. The presentation form in EXIST is for humans.

Numbers are shown as literals.

Identity

EXIST



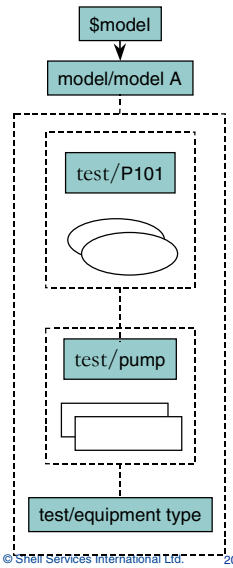
Definition: signs that refer to the same thing, e.g. [Test/A==Test/B].

This is used e.g. in showing a name in one model refers to the same object as a name in another model.

Exist also allows a graphical sign to be repeated for diagramming convenience. The two signs may be linked by a chain line (dash dot).

Models

EXIST



A Model is a set of EXIST statements. One model may be a subset of another model.

[$model: model/model A]

[model/model A == {

[test/equipment type == {test/pump, …}],

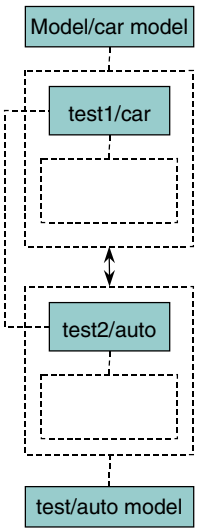
[test/pump == {test/P101, …}]

}]

Models may be combined to form a larger model using the union operator.

Mapping

EXIST



A mapping declares that the statements made by one model are equivalent to the statements made by another model. Equality and identity relations show how to map.

[test1/car model = test2/auto model]

[test1/car == test2/auto]

Issues

EXIST

- Is a Model a set of asserted propositions?